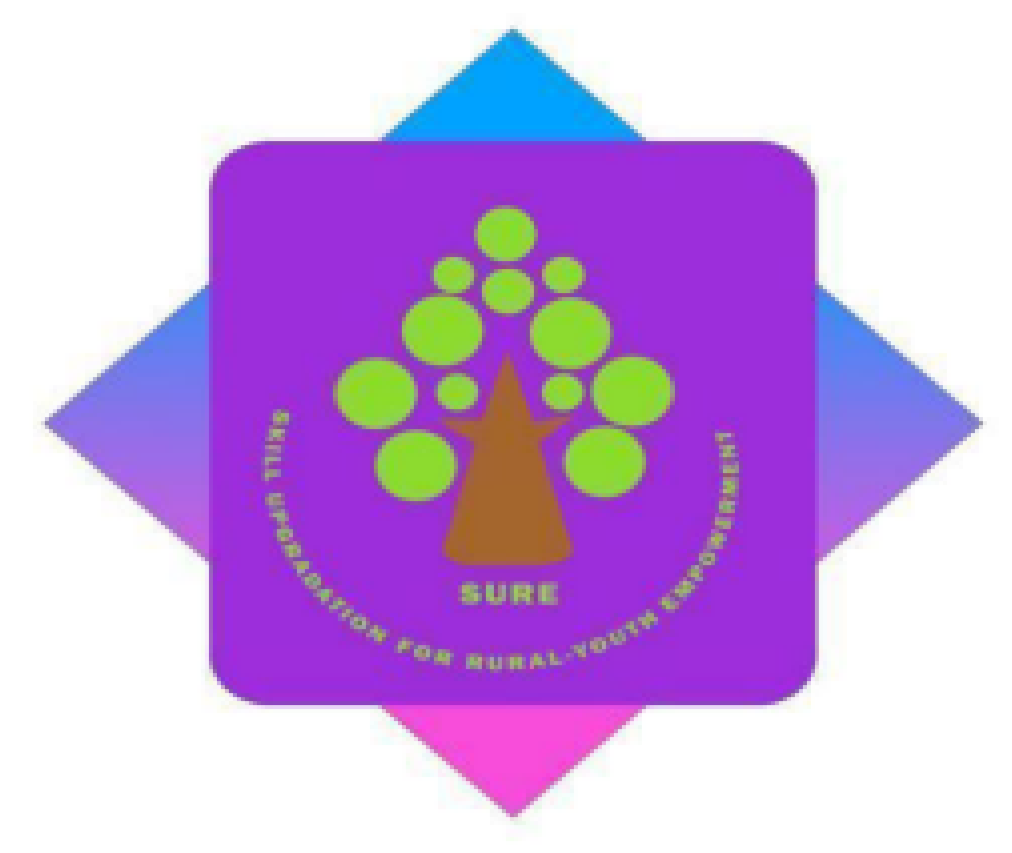
**ADVANCE VLSI DESIGNING**

**A CERTIFICATE COURSE CONDUCTED**

**BY**



**THE SURE TRUST**

**Skill Upgradation for Rural-youth Empowerment – TRUST**

**(www.suretrustforruralyouth.com)**

**COURSE TRAINING ATTENDED**

**BY**

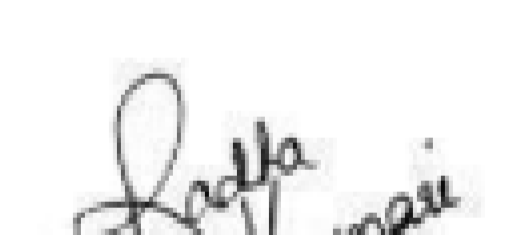
**Kondala Tarunkumar**

**September 2022 - December 2022**



**Declaration**

This is to certify that K.Tarunkumar has successfully completed the four months training given in “Advance VLSI Designing conducted by The SURE TRUST” during the period from octomber 2022 to January2023



Prof.Ch. Radha Kumari

Trainer

Executive Director & Founder

Ms. Kritika Ram



Mr

s. V

andana

|  |  |
| --- | --- |
| Trustee & Advisory  SURE TRUST | Director & Co-Founder SURE TRUST |

**TABLE OF CONTENTS**

1. The SURE TRUST
2. Course Content
3. Conduct of the Course

Student byelaws

Written Tests

Assignments

1. Student Feedback
2. Uniqueness of the Course according to student
3. Concluding Remarks

**1. Introduction to the SURE TRUST**

**Introduction to The SURE TRUST**

The SURE TRUST is born to enhance the employability of educated unemployed rural youth. It is observed that there is a wide gap between the skills acquired by students from the academic institutions and the skills required by the industry to employ them. Employability enhancement is done through giving one on one training in emerging technologies, completely through online mode. The mission of the SURE TRUST is to bridge the gap between the skills acquired and the skills required by training them in the most emerging technologies such as Artificial Intelligence (AI), Python Program, Machine Learning (ML), Deep Learning (DL), Data Science & Data Analytics,

Blockchain Technology, Robotic Process Automation (RPA), Project Management, Excel for Business Application, Statistical tools & Applications, Spoken English and Business Communication etc., that will enhance their employability. After completion of four months training in the course, the trainees will get live projects from industries as internship activity to get experience in applying to real time situation what they have learnt during the course. These projects will give them hands on experience which is much sought after by the prospective industry employing them. Currently students from all over India are enrolling for various courses offered by the SURE TRUST. The SURE TRUST offers every course free of cost with no financial burden of any kind to students. This initiative is purely a service-oriented one aiming to guide the rural youth who are educated but unemployed due to lack of upgradation in their skill sets. The birth of SURE TRUST is a God given boon to rural youth who could reach great heights either in employment or in entrepreneurship once they receive the training offered followed by the company internship. Many companies are coming forward to join their hands with us by offering internship projects to hand hold and lead the rural youth in their career settlement.

**Vision of the SURE TRUST**

The vision of the SURE TRUST is to enhance the employability of educated unemployed youth, particularly living in rural areas, through skill upgradation, with no cost to the students.

**Mission of the SURE TRUST**

The mission is to bridge the gap between the skills acquired in the academic institutions and the skills required in industries as a pre-condition for employment.

**Functioning of the SURE TRUST**

There are three dedicated, committed, and hard-working women on the board of management of the SURE TRUST who will look into the various administrative and other matters relating to the enrolment of students, organizing trainers, entering into agreements with companies for getting live projects to students as internship programs, and so on. All the three women on the board are all the alumni from Sri Sathya Sai Institute of Higher Learning, Anantapur Campus, deemed to be a University. The women board is supported by five eminent advisories who are from different walks of life and have made outstanding mark in career in their respective fields.

For more details about SURE TRUST please visit the website www.suretrustforruralyouth.com

**2. Course Content**

The SURE TRUST conducts four months of training for every course on a uniform basis. A session spanning across one to one & half hour is taken by the trainers for every major course. Sessions are conducted to complete the predesigned course structure within a fixed time period. Course content is designed to suit the current requirement of the Industry and validated by industry experts. The course content of all these courses is so dynamic that any changed condition noticed in the industry will automatically get reflected in the content of the respective course.

As the course content is dynamic, the Following is the course content of the current course in Advance VLSI Designing:

**Advance VLSI Designing**

## Objective:

● The dynamic curriculum of the Advance VLSI Design and Verification course fits perfectly with the career aim of fresh engineering graduates and helps them to ‘future-proof’ themselves and remain relevant for the rapidly evolving Semiconductor technology space.

## Course Content:

* Module 1 - Introduction to VLSI design and verification.
* Module 2 - Introduction to RTL design using Verilog. Modeling methods.
* Module 3 - Procedural assignments and Continuous assignments. MUX design.
* Module 4 - Introduction to Testbench design using Verilog.
* Module 5 - Introduction to Finite State Machines. Design and Verification of

FSM.

* Module 6 - Final project - Router 1x3 Design and Verification.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3. Conduct of the course:**

1. Modalities for the conduct of all the courses are fixed by the SURE TRUST which are uniformly followed across the courses.

Mode of Training --- Online

∙ Period of Training --- Four months

∙ Sessions per week --- 3 to 6

∙ Length of the session --- 1 to 2 hours

∙ Tests to be taken --- 2 per month

∙ Assignments --- 2 per month

∙ Last 15 days --- Final practice and preparing the course report

1. Student Byelaws:

Students enrolling for the courses under SURE TRUST are strictly required to follow the following Byelaws set for them.

**Byelaws for students to become eligible for certificate at the end of the course**

I. Minimum Attendance:

Every student must put in a minimum of 85% attendance in attending the classes for getting the eligibility to receive the certificates.

II.. Two written tests are to be taken each month:

Since the objective of the certification program is to turn out well-qualified students from the respective courses, minimum two written tests are to be taken in each month for each course to ensure that the students are pulled along the expected line of standard.

1. Assignment submissions:

Ten exercises constituting one assignment for every two to three new functions/topics taught, resulting in minimum seven such assignments are to be submitted during the four months period.

1. Preparing the final course report in the prescribed format:

During the last fifteen days in the fourth month, students many be asked to consolidate and compile all the assignments submitted in a word document along with the other chapters which will constitute a course report for each student. This report will be the unique contribution a student carries from the trust to showcase the rigorous training he/she received during the four months period. Besides the report will stand as a testimony to the detailed learning a student has acquired in the chosen area. This will facilitate the industry in handpicking the required student for the job. V. External Viva-voce:

Every student has to successfully clear the external viva-voce arranged in their respective course.

VI. KYC norms:

Each student wishing to enrol for the course must submit a written letter saying that he/she will not drop from the course until its completion, which will also be signed by father / mother besides the student himself / herself. VII. Attend the full class:

All the students are expected to attend each class for full duration. Some students are observed moving out of classes after logging in which does not go well with the learning objective of students.

VIII. Ensure discipline in the group:

All the students are advised strictly to follow group etiquette and restrain from posting in the group any unethical messages or teasing messages or personal interactive messages. This group is purely created for academic purposes and hence only academic interactions should go on

**VLSI DESIGNING**

# Assignment 1

**Design a 4-bit Ripple carry adders.**

**RTL**

**Step1:- Create Half\_Adder**

module Half\_Adder(A,B,Sum,Cout); //Port Declaration input A,B; output Sum,Cout; always @(A,B) begin //procedural assign

Sum=A^B; //assign Sum=A^B;

Cout=A&B; //assign Cout=A&B;

end endmodule

**Step2:- Creating Full\_Adder Using Halfadders** module Full\_Adder(in1,in2,Cin,Sum\_FA,Cout\_FA); //port Declaration input in1,in2,Cin; output Sum\_FA,Cout\_FA; wire Sum\_HA1,Cout\_HA1,Cout\_HA2;

//Module instantiation(Submodule Declaration)

Half\_Adder HA1(in1,in2,Sum\_HA1,Cout\_HA1);

Half\_Adder HA2(Sum\_HA1,Cin,Sum\_FA,Cout\_HA2);

Endmodule

**Step3:- Creating 4bit Ripple Carry Adder Using Full Adder** module Ripple\_Carry\_Adder(X,Y,Sum,Carryout);

//Port Declaration input[3:0]X,Y; output[3:0]Sum; output Carryout; wire CarryOut\_FA1,CarryOut\_FA2,CarryOut\_FA3;

//Order Based instantiation

Full\_Adder FA1(X[0],Y[0],1’b0,Sum[0],CarryOut\_FA1);

Full\_Adder FA2(X[1],Y[1],CarryOut\_FA1,Sum[1],CarryOut\_FA2);

Full\_Adder FA3(X[2],Y[2],CarryOut\_FA2,Sum[2],CarryOut\_FA3); Full\_Adder FA4(X[3],Y[3],CarryOut\_FA3,Sum[3],Carryout); endmodule

**Test Bench** module Rippel\_Carry\_Adder\_tb(); reg [3:0]A\_tb; reg [3:0]B\_tb; wire [3:0]Sum\_tb; wire Carryout\_tb; integer i;

Ripple\_Carry\_Adder DUT(A\_tb,B\_tb,Sum\_tb,Carryout\_tb);

initial

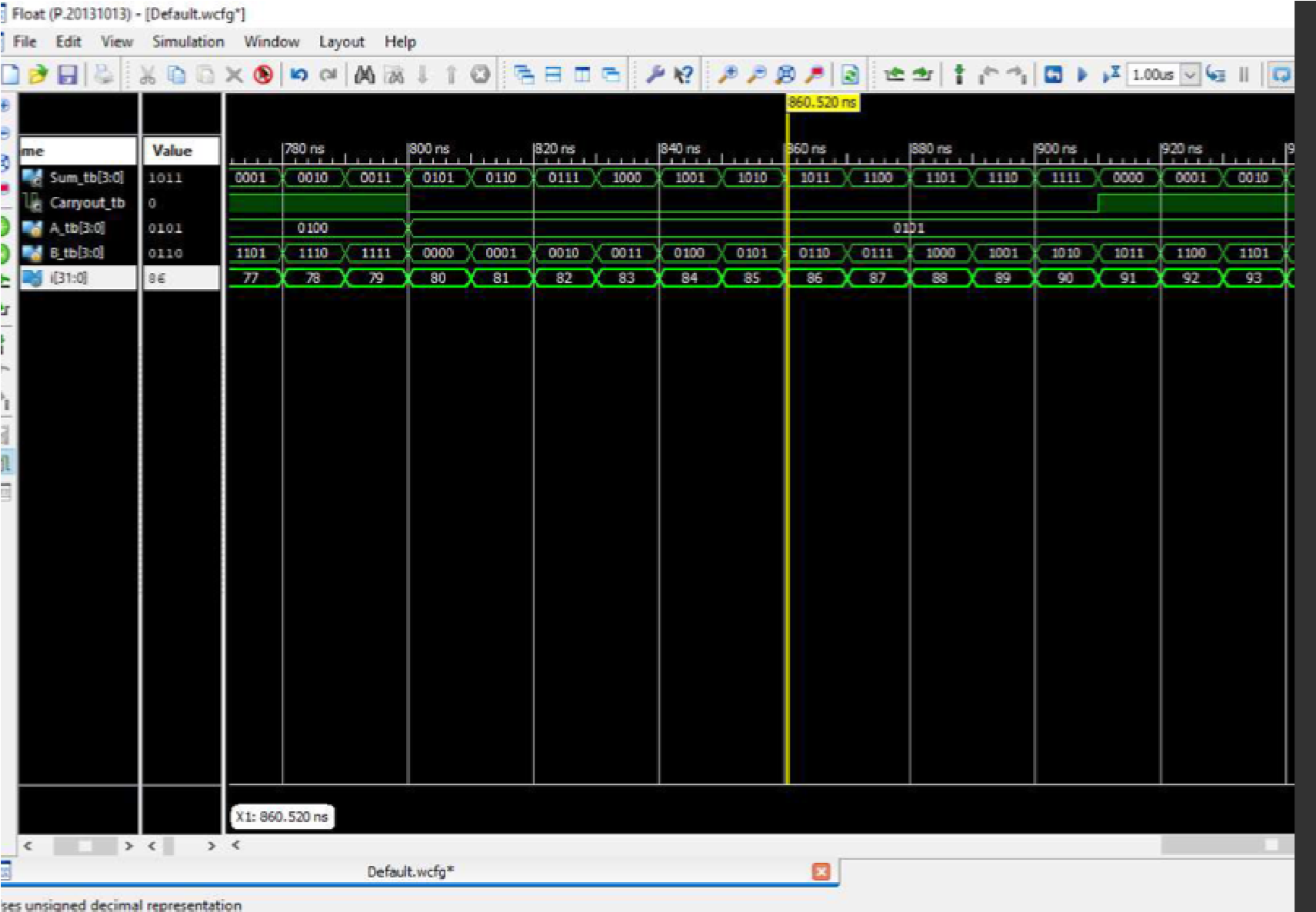
begin

for(i=0;i<512;i=i+1) begin

{ A\_tb,B\_tb}=i; #10; end

$finish; end endmodule

**Simulation**



# Assignment 2

## Design an 6x1 MUX and verify its functionality using its testbench.

## RTL

module MUX (I, S, Y);

input [7:0]I; input [2:0]S; output Y; reg Y;

always @(S,I)

begin

if (S == 3'b000)

Y = I[0]; else if (S == 3'b001)

Y = I[1]; else if (S == 3'b010)

Y = I[2]; else if (S == 3'b011)

Y = I[3]; else if (S == 3'b100)

Y = I[4]; else if (S == 3'b101)

Y = I[5]; else if (S == 3'b110)

Y = 1'bz; else if (S == 3'b111)

Y = 1'bz; end

endmodule

//assign Y = (S== 0)? I[0] : (S==1)? I[1] : (S==2)? I[2]: (S==3)? I[3];

**Testbench** module MUX\_tb();

reg [7:0]I; reg S[2:0];

wire Y;

//TEST BENCH OF 6:1 MUX MUX m(I,S,Y);

initial

begin

$dumpfile("Varun.vcd");

$dumpvars(0);

I=8'b00000000;

#10 I[0]=8'b00000001; {S[0],S[1],S[2]}= 3'b000;

#10 I[1]=8'b00000010; {S[0],S[1],S[2]} = 3'b001;

#10 I[2]=8'b00000100; {S[0],S[1],S[2]} = 3'b010; #10 I[3]=8'b00001000; {S[0],S[1],S[2]} = 3'b011 ;

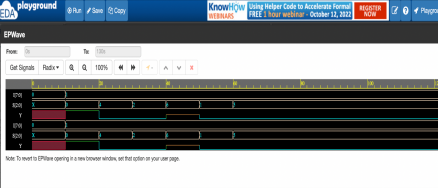
#10 I[4]=8'b00010000; {S[0],S[1],S[2]} = 3'b100;

#10 I[5]=8'b00100000; {S[0],S[1],S[2]} = 3'b101;

#70 $finish; end

endmodule

## Simulation



# Assignment 3

## 1) Bidirectional Buffer

## RTL

module Bibuffer (a, b, c); input c; inout a,b;

assign b = (c == 1)?a:1'bz; assign a = (c == 1)?b:1'bz; endmodule

## Testbench

module Bibuffer\_tb()

wire a, b; //we dont use wire as inputs in test bench reg ta,tb; integer i;

Bibuffer bi (a, b, c);

assign b = (c==1)?ta:1'bz; assign a = (c==0)?tb:1'bz;

initial begin

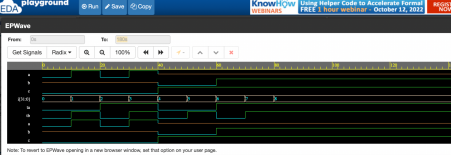
{ c, ta, tb} = 0;

for (i = 0; i<8; i+1)

{

{ c,ta,tb} = i;

} end endmodule **Simulation**



## 2) D flip Flop

## RTL

module D\_FF(d, clk, rst, q, qb); input rst, d, clk; output reg q, qb; always @(posedge clk)

begin

if (rst)

q <= 0;

else q <= d;

end

assign qb = ~q; endmodule

## Testbench

module D\_FF\_tb();

reg rst, clk, d; wire q, qb; D\_FF D1(d, clk, rst, q, qb); task Dff1(input a)

begin

@(posedge clk)

D = a; end

task Dff2(input v) begin @(posedge clk)

rst = v;

end

always

begin

#5 clk = ~clk; end initial

begin d = 0; rst = 0; clk = 0; Dff1(0);

Dff2(0);

Dff1(1);

Dff1(0);

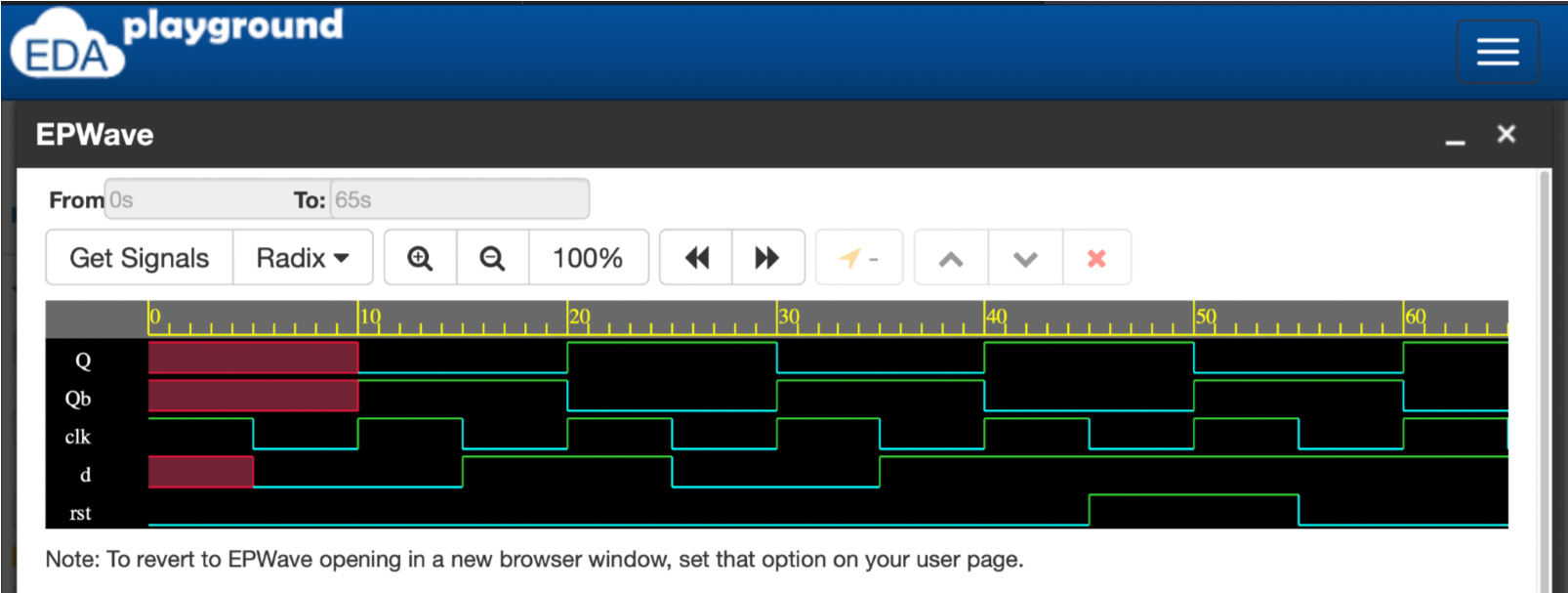
Dff2(1);

#50 $finish;

end

endmodule

## Simulation



## 3) J K Flip flop

## RTL

module jkff (Q, Qb, J, K, clk, rst); input rst, J, K, clk; output reg Q, Qb; always @(posedge clk)

begin

if (rst)

Q <= 0;

else if(J == 1'b0 && K == 1'b0) Q <= Q;

else if(J == 1'b0 && K == 1'b1) Q <= 0;

else if(J == 1'b1 && K == 1'b0) Q <= 1;

else if(J == 1'b1 && K == 1'b1) Q <= ~Q; Qb = ~Q; end

endmodule

## Testbench

module jkff\_test(); reg J,K,clk,rst; wire Q;

jk\_ff DUT(Q, Qb, J, K, clk, rst);

always #5 clk=~clk; tasK jK\_in( input [1:0]i); begin

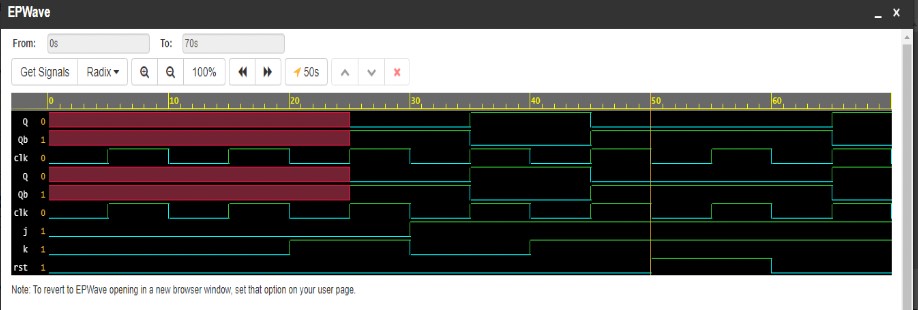
@(negedge clk)

{ J,K}=i; end endtask task reset(input a); begin @(negedge clk) rst=a;

end endtask

initial begin rst=0; J=0; K=0; clk=0; #5; jK\_in(0); jK\_in(1); jK\_in(2); jK\_in(3); reset(0); reset(1); end endmodule

## Simulation



**Assignment 4**

**1) Design and Verify a Mod 10 counter with Load functionality**

**RTL**

module Counter (Q, load, load\_en, pwr, clk, rst);

output reg Q; input load, load\_en, pwr, clk, rst; always@(posedge clk)

begin

if(~pwr)

Q <= 0; else

begin

if(rst)

Q <= 0; else if(load\_en) begin

if(load>1001)

Q = Q + 1; else

Q = load; end

else if(Q = 4'b1001)

Q <= 0; else

Q <= Q + 1; end

end

endmodule

**Testbench**

module Counter\_tb(); reg load, load\_en, pwr, clk, rst; wire Q;

Counter c1(Q, load, load\_en, pwr, clk, rst);

task initialize; begin clk = 0; rst = 0; pwr = 0; load\_en = 0; load = 0;

end

endtask task power (input a); begin @(negedge clk) pwr = a;

end

endtask

task reset;

begin

@(negedgeclk) rst=1;

@(negedgeclk) rst=0;

end

endtask task load\_yes(input[3:0]c);

begin @(negedgeclk) load\_en = 1; load = c;

@(negedge clk) load\_en = 0;

end

endtask

initial

begin

$dumpfile("jeans.vcd");

$dumpvars(0);

initialize; power(1);

#10; reset;

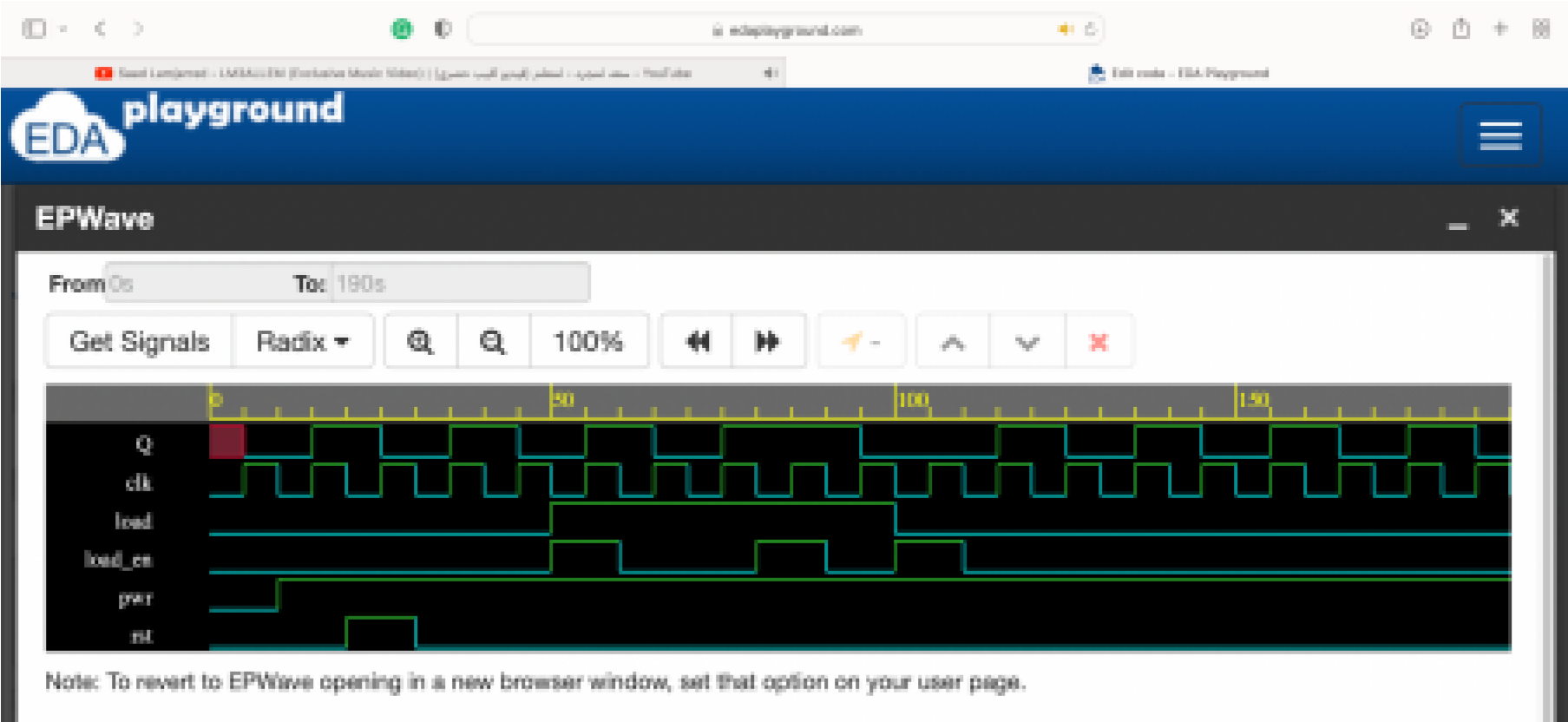
#20; load\_yes(4'b0011) ; #20; load\_yes(4'b1111) ;

#10; load\_yes(4'b1010);

#80$finish end

endmodule

## Simulation



## 2) 2-bit Asynchronous counter using Structural modeling

## RTL

module tff (q, qb, clk, rst, t);

input t, clk, rst; output reg q; output qb; always @(posedge clk)

begin

if (rst)

q <= 0;

else

begin

if(t == 1)

q <= ~q;

else q <= q;

end

assign qb = ~q; endmodule

module asy2 (out, clk, rst); input clk, rst; output [1:0]out;

wire [1:0]w; tff t1 (out[0], w[0], clk, rst, 1'b1); tff t2 (out[1],w[1],w[0], rst, 1'b1); endmodule

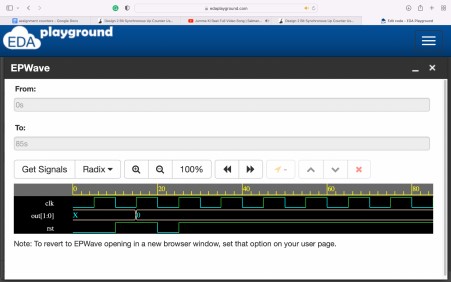
## Testbench

module asy2\_tb(); reg clk, rst; wire [1:0]out; wire [1:0]w; asy2 (out,clk,rst); always #5 clk = ~clk; initial begin $dumpfile("rcb.vcd"); $dumpvars(0); clk = 0; rst = 0; #10 rst = 1;

#10 rst = 0;

#5 rst = 1; #60 $finish; end endmodule

## Simulation



# Assignment 5

## 1) Single PORT RAM

## RTL

module memport (data, read\_en, write\_en, clk, addr);

input clk, write\_en, read\_en; input [3:0] addr; inout [7:0] data; reg [7:0] temp; reg [7:0] Mem [15:0]; assign data = (read\_en && !write\_en)?temp:8'bz; always@(posedge clk)

begin if(write\_en && !read\_en)

Mem[addr] <= data; else if(read\_en && !write\_en) temp <= Mem[addr];

end

endmodule

**Testbench** module memport\_tb();

reg clk, write\_en, read\_en; reg [3:0] addr;

wire [7:0] data; //no newed to use memory in test bench

memport m1 (data, read\_en, write\_en, clk, addr); integer i, j; reg [7:0] temp; assign data = (write\_en && !read\_en)?temp:8'bzzzz; always

#5 clk = ~clk;

task initialize; begin

write\_en = 0;

read\_en = 0; addr = 0; data = 0; clk = 0;

end

endtask

task write (input [3:0] a, input [7:0] dat);

begin @(negedge clk) write\_en = 1; read\_en = 0; addr = dat;

end

endtask

task read (input [3:0] b); begin @(negedge clk) write\_en = 0; read\_en = 1; addr = b;

end

endtask

initial

begin

initialize; for(i = 0; i < 16; i = i + 1) begin write(i, $random);

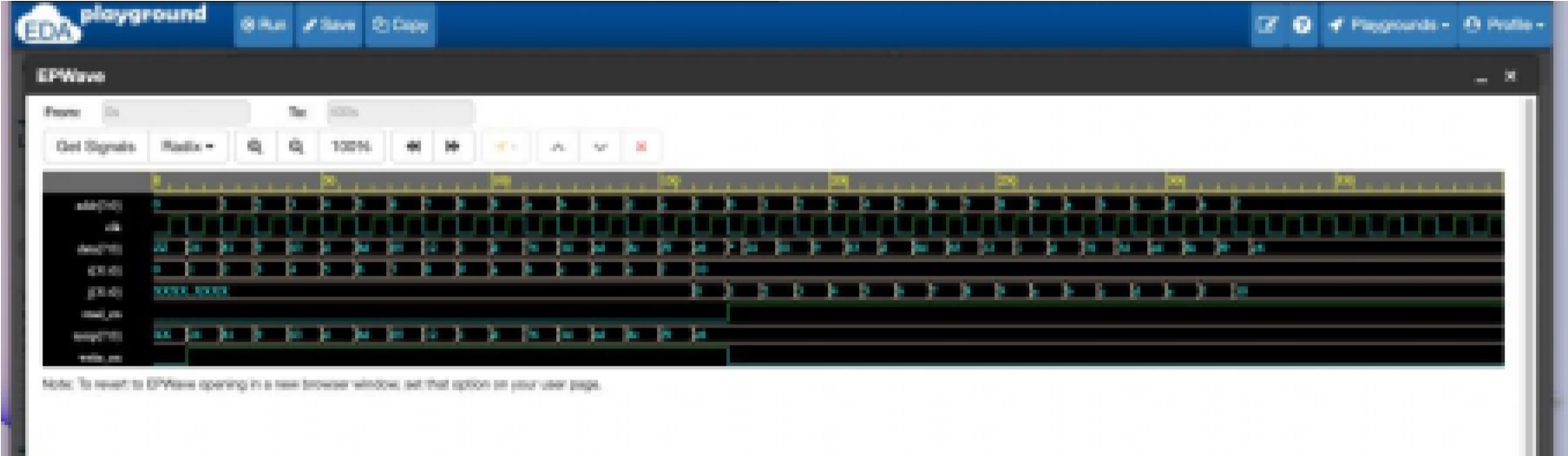
end

for(j = 0; j < 16; j = j + 1)

begin

read(j); end

end endmodule **Simulation**



## 2) Dual Port RAM

## RTL

module memport (data, read\_en, write\_en, clk, addr);

input clk, write\_en, read\_en; input [3:0] addr; inout [7:0] data; reg [7:0] temp; reg [7:0] Mem [15:0]; assign data = (read\_en && !write\_en)?temp:8'bz; always@(posedge clk)

begin

if(write\_en && !read\_en)

Mem[addr] <= data; else if(read\_en && !write\_en)

temp <= Mem[addr];

end

endmodule

**Testbench** module memport\_tb(); reg clk, write\_en, read\_en; reg [3:0] addr; wire [7:0] data; //no newed to use memory in test bench memport m1 (data, read\_en, write\_en, clk, addr); integer i, j; reg [7:0] temp; assign data = (write\_en && !read\_en)?temp:8'bzzzz; always

#5 clk = ~clk;

task initialize; begin

write\_en = 0; read\_en = 0; addr = 0; data = 0; clk = 0;

end

endtask

task write (input [3:0] a, input [7:0] dat);

begin @(negedge clk) write\_en = 1; read\_en = 0; addr = dat;

end

endtask

task read (input [3:0] b); begin @(negedge clk) write\_en = 0; read\_en = 1; addr = b;

end

endtask

initial

begin

initialize; for(i = 0; i < 16; i = i + 1) begin write(i, $random);

end

for(j = 0; j < 16; j = j + 1)

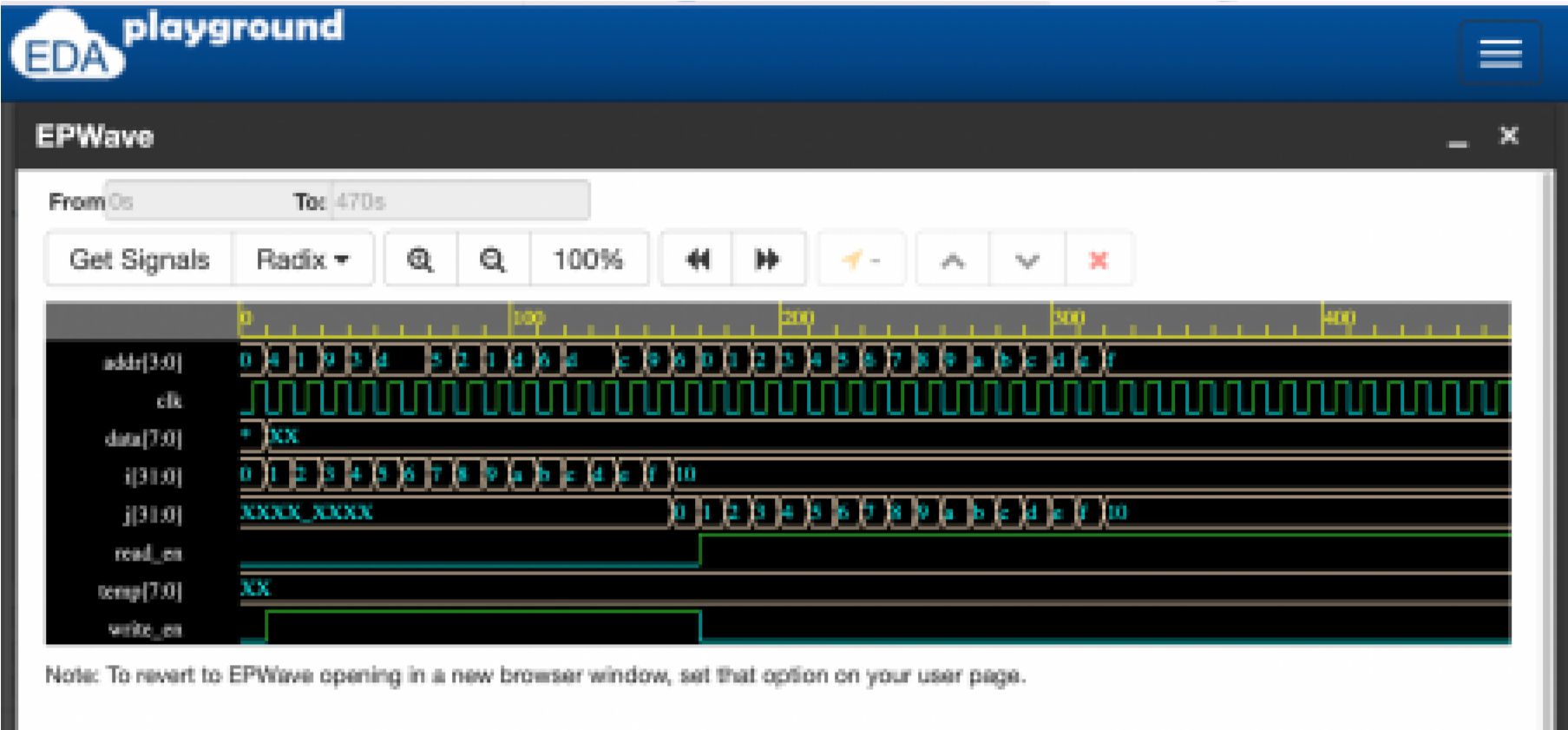
begin

read(j); end

end

endmodule

## Simulation



## 3) True Dual Port RAM

## RTL

module Lovedp2 (clk\_0, clk\_1, read\_0, read\_1, write\_1, write\_0, addr1, addr2, data\_0, data\_1);

input clk\_0, clk\_1, read\_0, read\_1, write\_0, write\_1;

input [3:0]addr1; input [3:0]addr2; inout [7:0]data\_0; inout [7:0]data\_1; reg [7:0]temp0; reg [7:0]temp1; reg [7:0] MEM [15:0];

assign data\_0 = (!write\_0 && read\_0)?temp0:8'bzzz; assign data\_1 = (!write\_1 && read\_1)?temp1:8'bzzz;

always@(posedge clk\_0) begin

if(!read\_0 && write\_0)

MEM[addr0] <= data\_0; else if(!write\_0 && read\_0)

temp0 <= MEM[addr0];

end

always@(negedge clk\_1) begin

if(!read\_1 && write\_1)

MEM[addr1] <= data\_1; else if(!write\_1 && read\_1)

temp1 <= MEM[addr1];

end endmodule

**Testbench** module Lovedp2\_test(); reg clk\_0, clk\_1, read\_0, read\_1, write\_0, write\_1; reg [3:0]addr1; reg [3:0]addr2;

wire [7:0]data\_0; wire [7:0]data\_1; reg [7:0]temp0; reg [7:0]temp1; integer i;

Lovedp2 vz(clk\_0, clk\_1, read\_0, read\_1, write\_1, write\_0, addr1, addr2, data\_0, data\_1);

always #5 clk\_0 = ~clk\_0; always #5 clk\_1 = ~clk\_1;

task initialize; begin

clk\_0 = 0; clk\_1 = 0; addr2 = 0; addr1 = 0; read\_0 = 0; read\_1 = 0; write\_0 = 0; write\_1 = 0; temp0 = 0; temp1 = 0;

end

endtask

task wr2;

begin

@(negedge clk\_0) write\_0 = 1;

read\_0 = 0;

@(posedge clk\_1) write\_1 = 1; read\_0 = 0;

end

endtask

task re2;

begin

@(negedge clk\_0) write\_0 = 0; read\_0 = 1;

@(posedge clk\_1) write\_1 = 0; read\_1 = 1;

end

endtask

task brewr;

begin

@(negedge clk\_0) write\_0 = 1; read\_0 = 0;

@(posedge clk\_1) write\_1 = 0; read\_1 = 1;

end

task stimulus\_0 (input [3:0]adr, input [7:0]da);

begin @(negedge clk\_0) addr1 <= adr; temp0 <= da;

end

endtask

task stimulus\_1 (input [3:0]a0, input [7:0]d0);

begin @(posedge clk\_1) addr2 = a0; temp1 = d0;

end

endtask

initial

begin

$dumpfile("varunn.vcd");

$dumpvars(0); initialize; wr2;

end initial

fork

#5 for(i = 0; i < 7; i = i + 1) stimulus\_0(i, $random);

#5 for(i = 8; i < 16; i = i + 1) stimulus\_1(i, $random);

join

initial

begin

initialize; #200 re2; end initial

fork

#200 for(i = 0; i < 7; i = i + 1) stimulus\_0(i, $random);

#200 for(i = 8; i < 16; i = i + 1) stimulus\_1(i, $random);

join

initial

begin

initialize;

#410 brewr; end initial

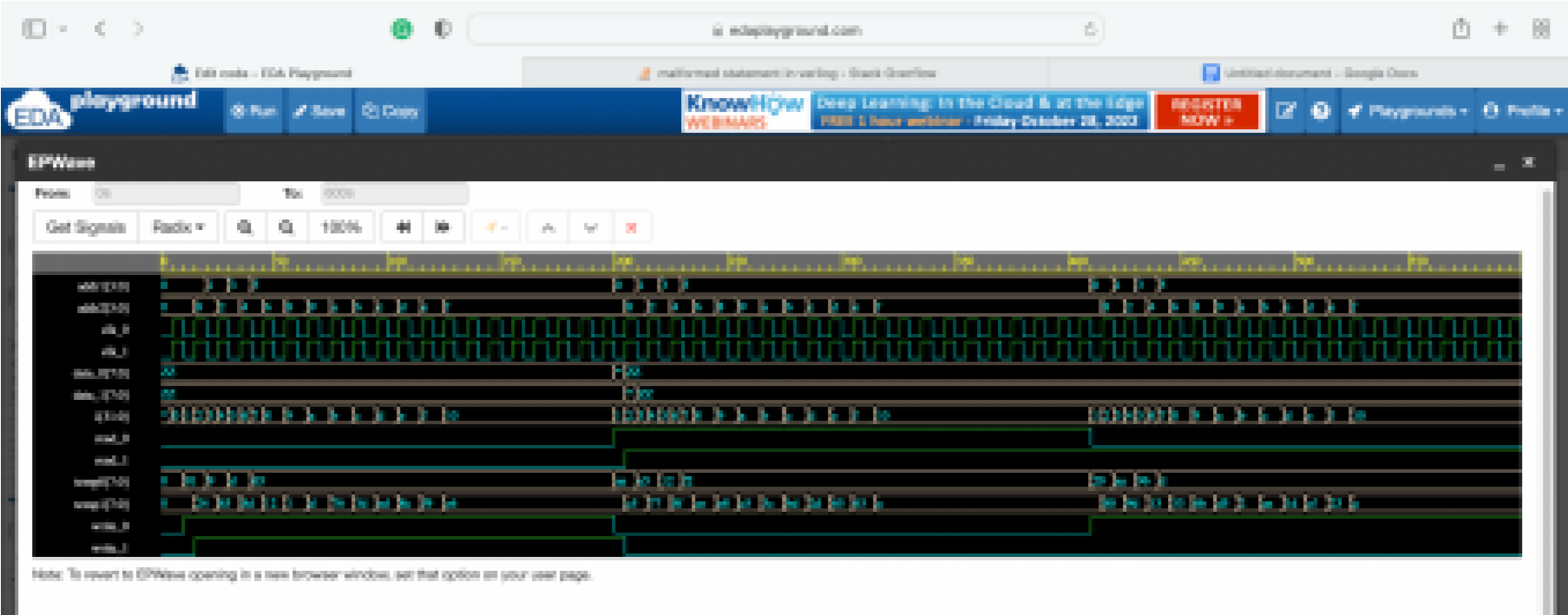
fork

#410 for(i = 0; i < 7; i = i + 1) stimulus\_0(i, $random);

#410 for(i = 8; i < 16; i = i + 1) stimulus\_1(i, $random);

join

## endmodule Simulation:



# Assignment 6

## FIFO Module

## RTL

module FIFO (clk, rst, din, dout, read\_en, write\_en, full, empty); parameter width = 8; parameter depth = 16; parameter addr\_width = 4; output full, empty; output [width - 1:0]dout; input [width - 1:0]din; input clk, read\_en, write\_en, rst; reg [addr\_width - 1:0]write\_pointer; reg [addr\_width - 1:0]read\_pointer; reg [addr\_width : 0]status\_counter = 0; reg [width - 1:0] MEM [depth - 1:0];

integer i;

always@(posedge clk) begin if(write\_en && !full && !read\_en) status\_counter <= status\_counter + 1;

else if (!write\_en && !empty && read\_en) status\_counter <= status\_counter - 1;

else if(write\_en && !full) && (read\_en && !empty) status\_counter <= status\_counter;

end

assign full = (status\_counter == depth)?1:0; assign empty = (status\_counter == 5'b0000)?1:0;

always@(posedge clk) begin

if(rst) write\_pointer <= 0;

else if(write\_en) write\_pointer <= write\_pointer + 1;

end always@(posedge clk) begin

if(rst)

read\_pointer <= 0;

else if(read\_en) read\_pointer <= read\_pointer + 1; end

always@(posedge clk) begin

if(rst)

begin for(i = 0; i<depth; i = i+1)

MEM [i] <= 0; end

else if(write\_en)

MEM [write\_pointer] <= data\_in; else if (read\_en) data\_out <= MEM [read\_pointer]; end endmodule

## Testbench

module FIFO\_tb(); reg [width - 1:0]din; reg clk, read\_en, write\_en, rst; parameter width = 8; parameter depth = 16; parameter addr\_width = 4; wire full, empty; wire [width - 1:0]dout; reg [addr\_width - 1:0]write\_pointer; reg [addr\_width - 1:0]read\_pointer; reg [addr\_width : 0]status\_counter = 0;

FIFO fanta (clk, rst, din, dout, read\_en, write\_en, full, empty); always

#5 clk = ~clk;

task initialize; begin clk = 0; read\_en = 0; write\_en = 0; rst = 0; din = 8'd0; end endtask task reset; begin

@(negedge clk) rst <= 1;

@(negedge clk) rst <= 0; end endtask

task write(input [width - 1:0] d, input [addr\_width - 1:0]a);

begin

@(negedge clk) din <= d;

end endtask

initial begin

$dumpfile("fantaa.vcd");

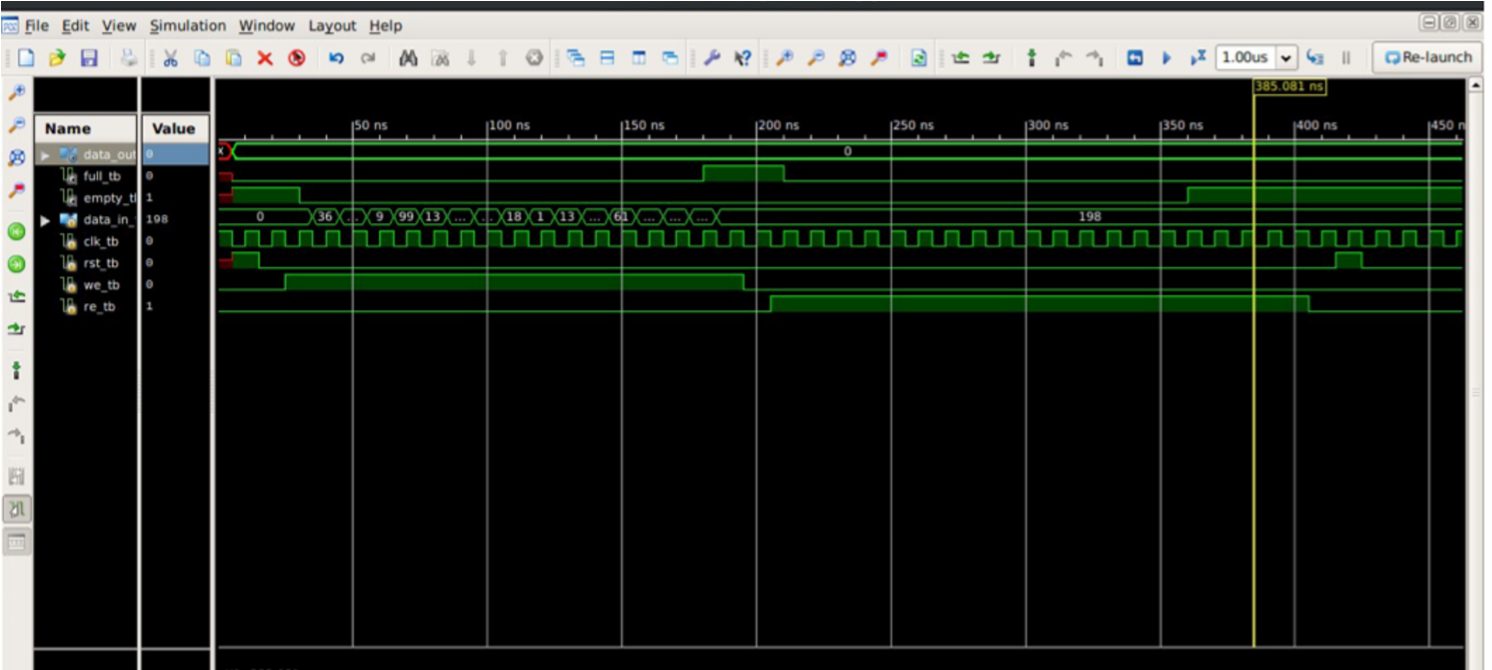
$dumpvars(0);

initialize; reset; #15; write\_en = 1; for(i= 0; i < depth; i = i + 1) write($random); #20

write\_en = 0; read\_en = 1;

#200 $finish; end endmodule

## Simulation



**Assignment 7**

## 1) A sequential circuit has two inputs and two outputs. The inputs (X1X2) represent a 2- bit binary number, N. If the present value of N is greater than the previous value, then Z1=1. If the present value of N is less than the previous value, then Z2 =1. Otherwise Z1, Z2 are 0. Draw FSM for the same.

## RTL

module \_2in2out (X, Z, clk, rst); input [1:0]X; output reg [1:0]Z; input clk, rst;

reg [2:0] current\_state, next\_state;

parameter IDLE = 3'b00, N00 = 3'b001, N01 = 3'b010, N10 = 3'b011,N11 = 3'b100; always@(posedge clk) begin if(rst)

current\_state <= IDLE;

else current\_state <= next\_state;

end

always@(X, current\_state) begin case(current\_state) IDLE: if(X == 2'b00) begin next\_state <= N00;

end

else if(X == 2'b01)

begin next\_state <= N01;

Z <= 2'b00; end

else if(X == 2'b10)

begin next\_state <= N10;

Z <= 2'b00; end

else if(X == 2'b11)

begin next\_state <= N11 ;

Z <= 2'b00;

end

N00: if(X == 2'b00) begin

next\_state <= N00;

Z <= 2'b00; end else if(X == 2'b01) begin

next\_state <= N01;

Z <= 2'b10; end else if(X == 2'b10) begin

next\_state <= N10;

Z <= 2'b10; end else if(X == 2'b11) begin

next\_state <= N11 ;

Z <= 2'b10; end

N01: if(X == 2'b00) begin

next\_state <= N00;

Z <= 2'b01; end else if(X == 2'b01) begin

next\_state <= N01;

Z <= 2'b00; end else if(X == 2'b10) begin

next\_state <= N10;

Z <= 2'b10; end else if(X == 2'b11) begin

next\_state <= N11 ;

Z <= 2'b10; end

N10: if(X == 2'b00) begin

next\_state <= N00;

Z <= 2'b01; end else if(X == 2'b01) begin next\_state <= N01;

Z <= 2'b01; end else if(X == 2'b10) begin next\_state <= N10;

Z <= 2'b00; end else if(X == 2'b11) begin next\_state <= N11 ;

Z <= 2'b10; end

N11: if(X == 2'b 00) begin next\_state <= N00;

Z <= 2'b01; end else if(X == 2'b01) begin next\_state <= N01;

Z <= 2'b01; end else if(X == 2'b10) begin next\_state <= N10;

Z <= 2'b01; end else if(X == 2'b11) begin next\_state <= N11 ;

Z <= 2'b00; end endcase end endmodule

**Testbench** module \_2in2out\_tb();

reg [1:0]X; wire [1:0]Z; reg clk, rst; integer i; \_2in2out e(X, Z, clk, rst); always

#5 clk = ~clk; task initialize; begin clk = 0;

rst = 0;

end endtask task data (input [1:0] Xi); begin

@(posedge clk)

X <= Xi; end endtask

initial begin

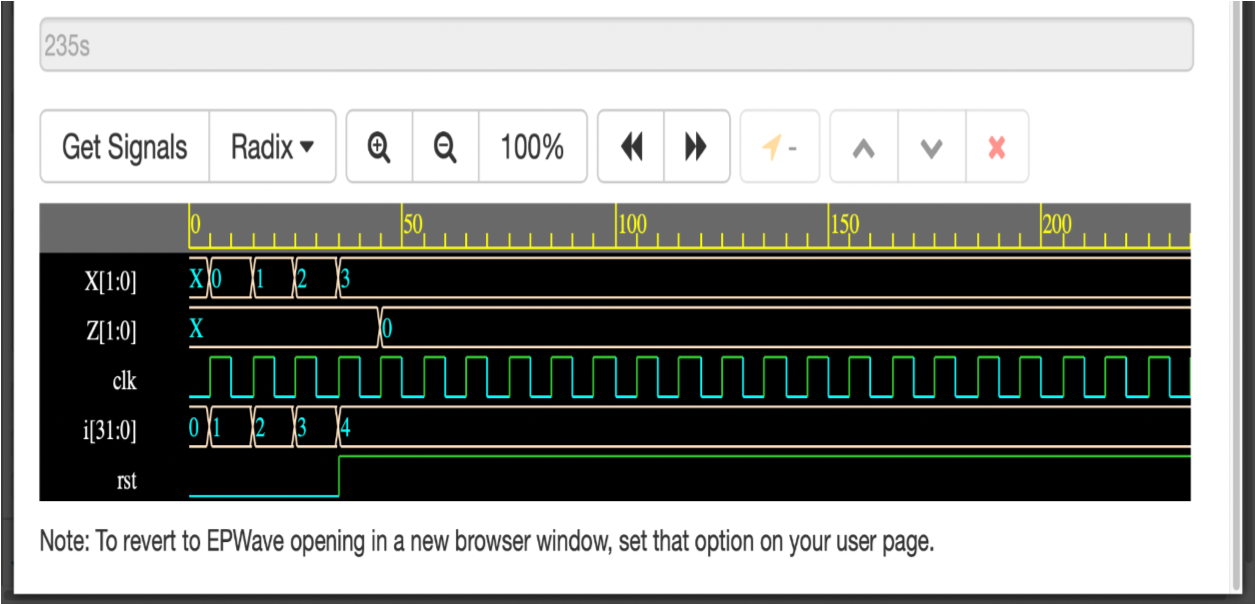
$dumpfile("mine.vcd");

$dumpvars(0);

initialize; rst = 1'b0; for(i = 0; i <= 3; i = i + 1) data(i); rst = 1'b1;

#200 $finish; end endmodule

## Simulation



## 2) Design an FSM that has 1 input and 1 output. The o/p becomes 1 and remains 1 when at least two 0s and two 1s have occurred as i/p's

## RTL

module \_1in1out(in, out, clk, rst); output out; input in, clk, rst;

reg [2:0] current\_state, next\_state;

parameter IDLE = 3'b000, S0 = 3'b001, S1 = 3'b010, S10 = 3'b011, MAX0 = 3 'b100, MAX1 = 3'b101; always@(posedge clk) begin if(rst)

current\_state <= IDLE;

else

current\_state <= next\_state;

end always@(in) begin

case(current\_state)

IDLE: if(in == 1'b0)next\_state <= S0; else if(in == 1'b1)next\_state <= S1;

S0: if(in == 1'b0)next\_state <= MAX0; else if(in == 1'b1)next\_state <= S10;

S1: if(in == 1'b0)next\_state <= S10; else if(in == 1'b1)next\_state <= MAX1;

S10: if(in == 1'b0)next\_state <= MAX0; else if(in == 1'b1)next\_state <= MAX1;

MAX0: if(in == 1'b0)next\_state <= MAX0; else if(in == 1'b1)next\_state <= MAX1;

MAX1: if(in == 1'b0)next\_state <= MAX0; else if(in == 1'b1)next\_state <= MAX1;

endcase end

assign out = (current\_state == IDLE)?1'b0:(current\_state == S0)?1'b0:(current\_state == S1)?1'b0:(current\_state == S10)?1'b0:(current\_state == MAX0 || MAX1)?1'b1:1'bzz; endmodule

## Testbench

module \_1in1out\_tb(); reg in;

wire out; reg clk, rst; integer i; \_1in1out e(in, out, clk, rst); always

#5 clk = ~clk; task initialize; begin clk = 0;

rst = 0;

end endtask task data (input Xi); begin

@(posedge clk) in <= Xi;

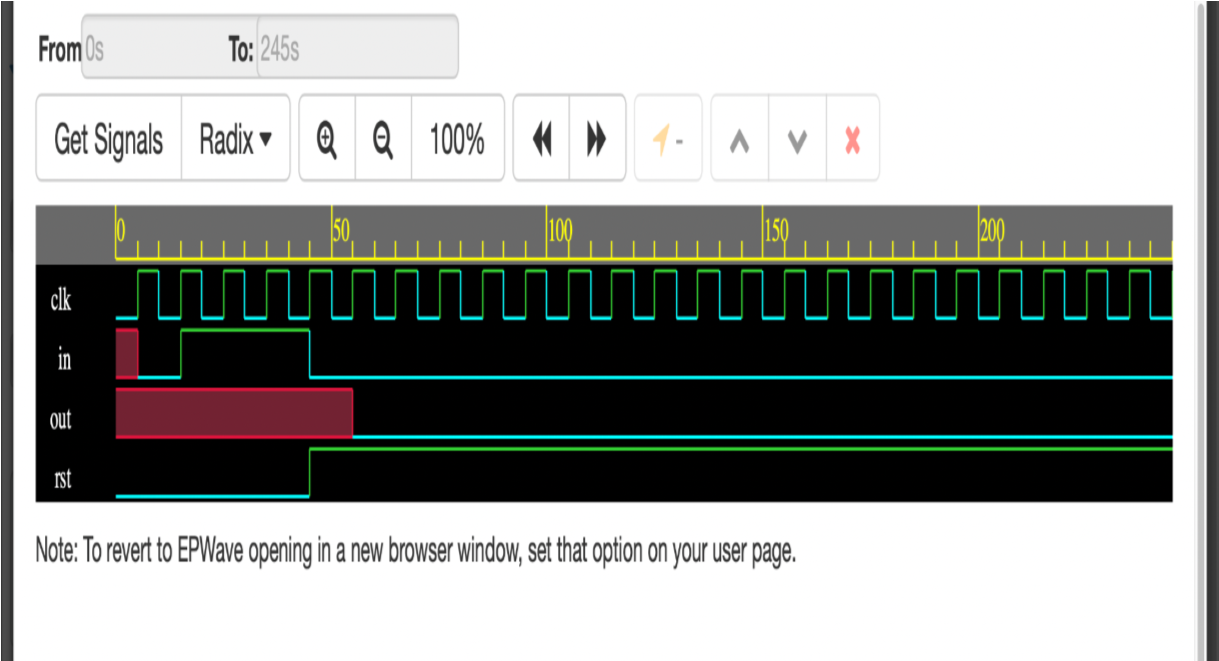
end endtask

initial begin

$dumpfile("mine.vcd");

$dumpvars(0);

initialize; rst = 1'b0; data(0); data(1); data(1); data(1); data(0); rst = 1'b1;

#200 $finish; end endmodule **Simulation:**

## FSM

module FSM (clk, rst, Din, Dout); input clk, rst; input [1:0] Din; output [1:0] Dout;

parameter IDLE = 3'b000, S0 = 3'b001, S1 = 3'b010, S2 = 3'b011, S3 = 3'b100; reg [2:0] current\_state, next\_state; always@(posedge clk) begin if(rst)

current\_state = IDLE;

else current\_state = next\_state;

end

always@(current\_state, Din) begin

case(current\_state)

IDLE : if(Din == 2'b00)next\_state <= S0; else if(Din == 2'b01)next\_state <= S1; else if(Din == 2'b10)next\_state <= S2;

else if(Din == 2'b11) next\_state <= S3;

S1: if(Din == 2'b00)next\_state <= S1;

else if(Din == 2'b01)next\_state <= S1; else if(Din == 2'b10)next\_state <= S2;

else if(Din == 2'b11) next\_state <= S3;

S2: if(Din == 2'b00)next\_state <= S2;

else if(Din == 2'b01)next\_state <= S2; else if(Din == 2'b10)next\_state <= S2;

else if(Din == 2'b11) next\_state <= S3;

S3: if(Din == 2'b00)next\_state <= S3;

else if(Din == 2'b01)next\_state <= S3; else if(Din == 2'b10)next\_state <= S3; else if(Din == 2'b11) next\_state <= S3;

default : next\_state <= 2'bzz;

endcase end

assign Dout = (current\_state == IDLE)2'b00:(current\_state == S1)2'b01:(current\_state

== S2)2'b10:(current\_state == S3)2'b11:2 'bzz; endmodule

## Testbench

module FSM tb();

reg clk, rst; reg [1:0] Din; wire [1:0] Dout; FSM F1(clk, rst, Din, Dout); always

#5 clk = ~clk; task data (input [1:0] D); begin

@(negedge clk)

Din <= D; end endtask

task reset (); begin

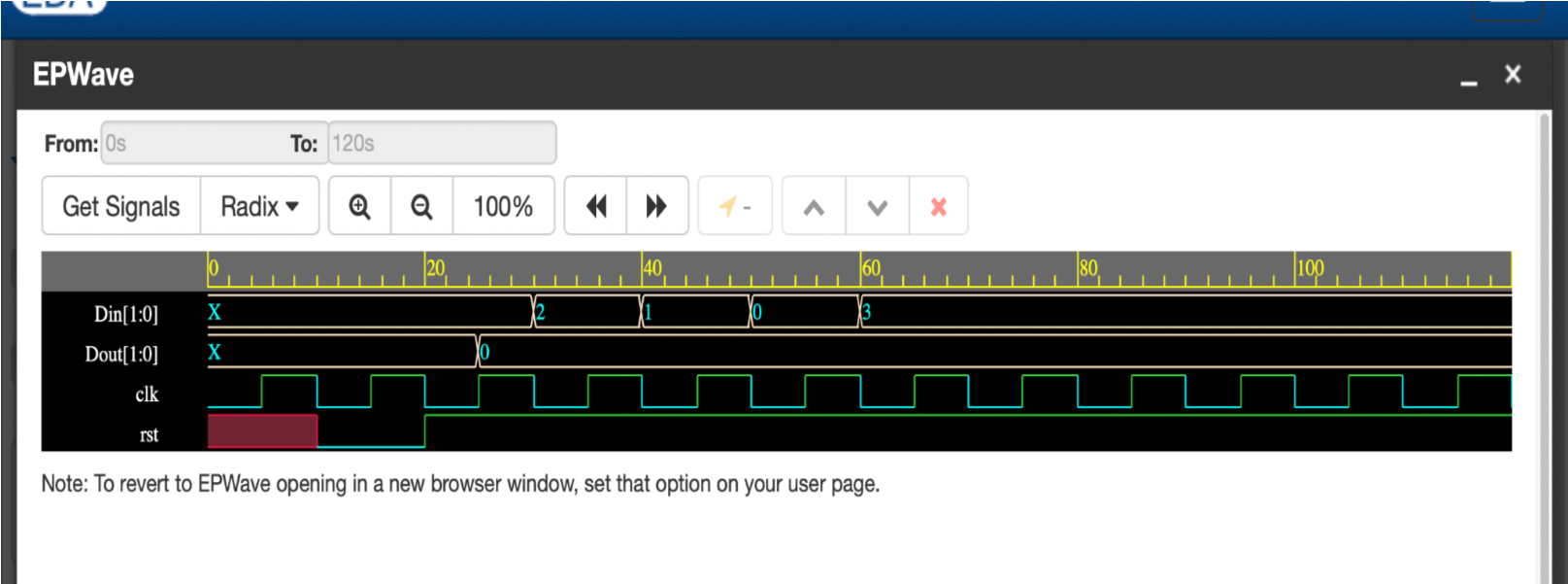
@(negedge clk) rst <= 0;

@(negedge clk) rst <= 1;

end endtask

initial begin

$dumpfile("varuuun.vcd"); $dumpvars(0); clk = 0; reset; data(2); data(1); data(4); data(3); data(23);

#50 $finish; end endmodule

## Simulation

**Final Project**

**ROUTER 3X1**

## Designing a Router using FIFO, FSM, Register and Synchronizer

## 1) FIFO RTL

module

router\_fifo(clk,resetn,soft\_reset,write\_enb,read\_enb,lfd\_state,datain,full,empty,dataout);

input clk,resetn,soft\_reset,write\_enb,read\_enb,lfd\_state; input [7:0]datain; output reg full,empty; output reg [7:0]dataout;

reg [3:0]read\_ptr,write\_ptr; reg [5:0]count; reg [8:0]fifo[15:0];

integer i; reg temp; reg [4:0] incrementer;

//lfd\_state always@(posedge clk)

begin

if(!resetn)

temp<=1'b0;

else temp<=lfd\_state; end

//Incrementer always @(posedge clk ) begin

if( !resetn )

incrementer <= 0;

else if( (!full && write\_enb) && ( !empty && read\_enb ) ) incrementer<= incrementer;

else if( !full && write\_enb ) incrementer <= incrementer + 1;

else if( !empty && read\_enb ) incrementer <= incrementer - 1;

else incrementer <= incrementer;

end

//Full and empty logic always @(incrementer) begin if(incrementer==0)

empty = 1 ; else empty = 0;

if(incrementer==4'b1111)

full = 1; else

full = 0; end

//Fifo write logic always@(posedge clk) begin

if(!resetn || soft\_reset) begin

for(i=0;i<16;i=i+1)

fifo[i]<=0;

end

else if(write\_enb && !full)

{ fifo[write\_ptr[3:0]][8],fifo[write\_ptr[3:0]][7:0]}<={temp,datain}; end

//FIFO READ logic always@(posedge clk)

begin

if(!resetn)

dataout<=8'd0;

else if(soft\_reset)

dataout<=8'bzz;

else

begin

if(read\_enb && !empty)

dataout<=fifo[read\_ptr[3:0]];

if(count==0) // COMPLETELY READ

dataout<=8'bz;

end

end

//counter logic always@(posedge clk)

begin

if(read\_enb && !empty)

begin

if(fifo[read\_ptr[3:0]][8]) count<=fifo[read\_ptr[3:0]][7:2]+1'b1; else if(count!=6'd0)

count<=count-1'b1; end end

//pointer logic always@(posedge clk) begin

if(!resetn || soft\_reset) begin

read\_ptr = 5'd0;

write\_ptr = 5'd0;

end

else

begin

if(write\_enb && !full) write\_ptr=write\_ptr+1'b1;

if(read\_enb && !empty)

read\_ptr=read\_ptr+1'b1;

end

end

endmodule

## Testbench

module router\_fifo\_tb();

//parameter DELAY=10;

//parameter DEPTH=16, WIDTH=9, ADD\_SIZE=5;

reg clk, resetn, write\_enb, read\_enb, lfd\_state, soft\_reset; reg [7:0]data\_in; wire full, empty; wire [7:0]data\_out;

router\_fifo DUT (.clk(clk),

.resetn(resetn),

.soft\_reset(soft\_reset),

.write\_enb(write\_enb),

.read\_enb(read\_enb),

.lfd\_state(lfd\_state),

.datain(data\_in),

.full(full),

.empty(empty),

.dataout(data\_out));

initial begin clk = 1; forever #5 clk=~clk; end

initial begin resetn=1'b0; #10; resetn=1'b1; soft\_reset=1'b0; lfd\_state=1'b1; write\_enb=1'b1; #10; repeat(17)

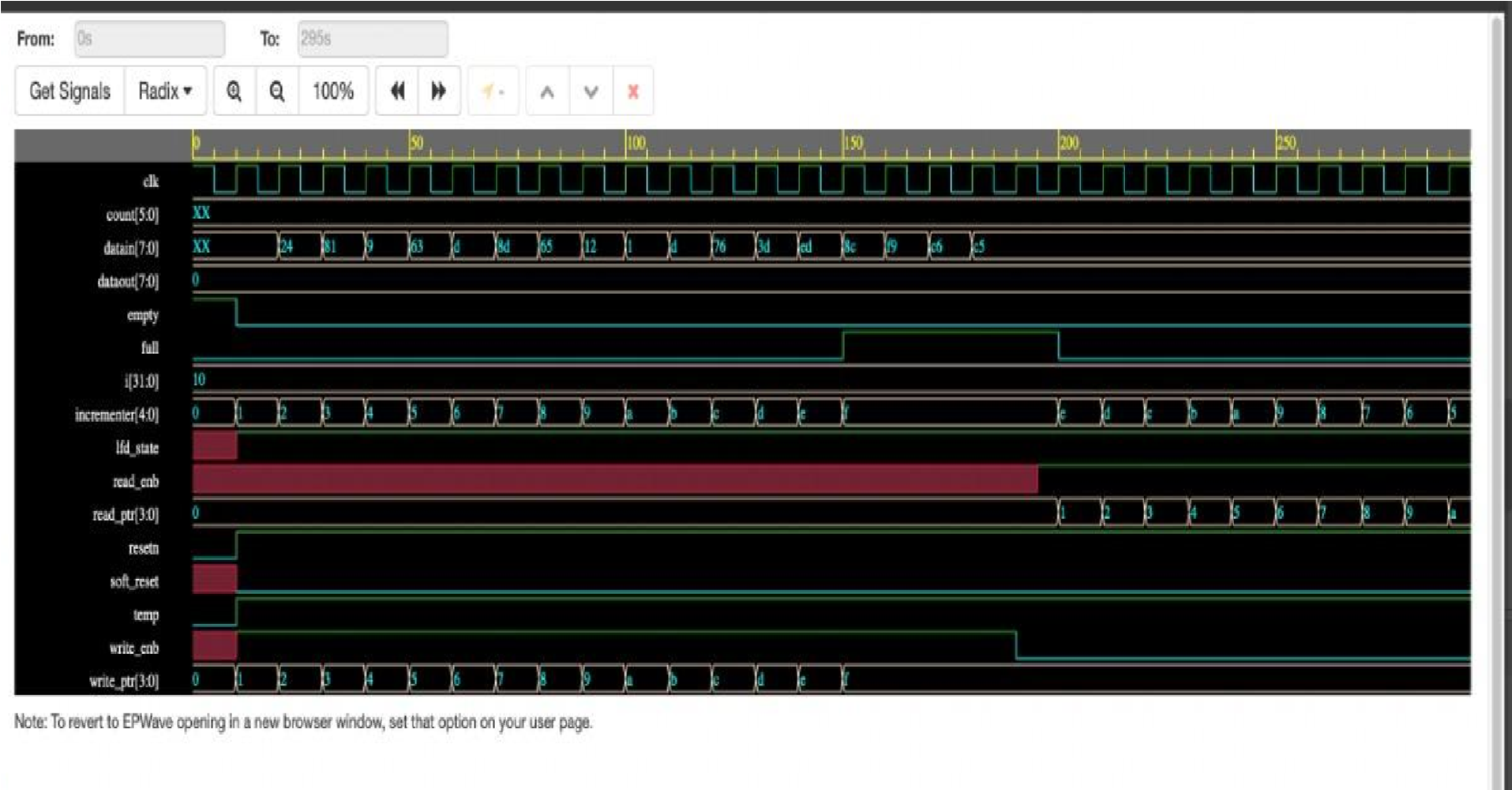
begin data\_in=$random%256;

#10; end

write\_enb=1'b0; #5; read\_enb=1'b1;

#100;

$finish; end endmodule **Simulation**



## 2) Synchronizer RTL

module router\_sync ( nput, clk, resetn, detect\_add, write\_enb\_reg,read\_enb\_0 i,read\_enb\_1,read\_enb\_2,empty\_0,empty\_1,empty\_2,full\_0,full\_1,full\_2) input [1:0]datain,

output wire vld\_out\_0,vld\_out\_1,vld\_out\_2, output reg [2:0]write\_enb,

output reg fifo\_full, soft\_reset\_0,soft\_reset\_1,soft\_reset\_2);

reg [1:0]temp; reg [4:0]count0,count1,count2;

always@(posedge clk) begin

if(!resetn) temp <= 2'd0;

else if(detect\_add) temp<=datain;

end

//for fifo full

always@(\*) begin case(temp)

2 'b00: fifo\_full=full\_0;

2 'b01: fifo\_full=full\_1; 2 'b10: fifo\_full=full\_2;

default: fifo\_full=0;

endcase

end

//write enable always@(\*) begin if(write\_enb\_reg) begin case(temp)

2 'b00: write\_enb=3'b001;

2 'b01: write\_enb=3'b010; 2 'b10: write\_enb=3'b100; default: write\_enb=3'b000;

endcase

end else write\_enb = 3'b000;

end

//valid out

assign vld\_out\_0 = !empty\_0; assign vld\_out\_1 = !empty\_1; assign vld\_out\_2 = !empty\_2;

//soft reset counter always@(posedge clk) begin

if(!resetn) count0<=5'b0;

else if(vld\_out\_0) begin if(!read\_enb\_0) begin if(count0==5'b11110) begin soft\_reset\_0<=1'b1; count0<=1'b0;

end

else begin count0<=count0+1'b1; soft\_reset\_0<=1'b0;

end

end

else count0<=5'd0;

end

else count0<=5'd0;

end

always@(posedge clk) begin

if(!resetn) count1<=5'b0;

else if(vld\_out\_1) begin if(!read\_enb\_1) begin if(count1==5'b11110) begin soft\_reset\_1<=1'b1; count1<=1'b0;

end

else begin count1<=count1+1'b1; soft\_reset\_1<=1'b0;

end

end

else count1<=5'd0;

end

else count1<=5'd0;

end

always@(posedge clk) begin

if(!resetn) count2<=5'b0;

else if(vld\_out\_2) begin if(!read\_enb\_2) begin if(count2==5'b11110) begin soft\_reset\_2<=1'b1; count2<=1'b0;

end

else begin count2<=count2+1'b1; soft\_reset\_2<=1'b0;

end

end

else count2<=5'd0;

end

else count2<=5'd0;

end

endmodule

## Testbench

module router\_sync\_tb();

reg clk, resetn, detect\_add, full\_0, full\_1, full\_2, empty\_0, empty\_1, empty\_2, write\_enb\_reg, read\_enb\_0, read\_enb\_1, read\_enb\_2;

reg [1:0]datain; wire [2:0]write\_enb;

wire fifo\_full, soft\_reset\_0, soft\_reset\_1, soft\_reset\_2, vld\_out\_1, vld\_out\_2, vld\_out\_0;

router\_sync DUT(.clk(clk),

.resetn(resetn), .detect\_add(detect\_add), .full\_0(full\_0), .full\_1(full\_1), .full\_2(full\_2),

.empty\_0(empty\_0), .empty\_1(empty\_1), .empty\_2(empty\_2),

.write\_enb\_reg(write\_enb\_reg), .read\_enb\_0(read\_enb\_0), .read\_enb\_1(read\_enb\_1),

.read\_enb\_2(read\_enb\_2), .datain(datain), .write\_enb(write\_enb), .fifo\_full(fifo\_full),

.soft\_reset\_0(soft\_reset\_0), .soft\_reset\_1(soft\_reset\_1), .soft\_reset\_2(soft\_reset\_2),

.vld\_out\_1(vld\_out\_1), .vld\_out\_2(vld\_out\_2), .vld\_out\_0(vld\_out\_0) );

//clock generation initial begin clk = 1;

forever

#5 clk=~clk; end

task reset; begin resetn=1'b0; #10;

resetn=1'b1;

end

endtask

task task1(); begin detect\_add=1'b1;

datain=2'b10; read\_enb\_0=1'b0; read\_enb\_1=1'b1; read\_enb\_2=1'b0; write\_enb\_reg=1'b1; full\_0=1'b0; full\_1=1'b1; full\_2=1'b1; empty\_0=1'b1; empty\_1=1'b0; empty\_2=1'b0;

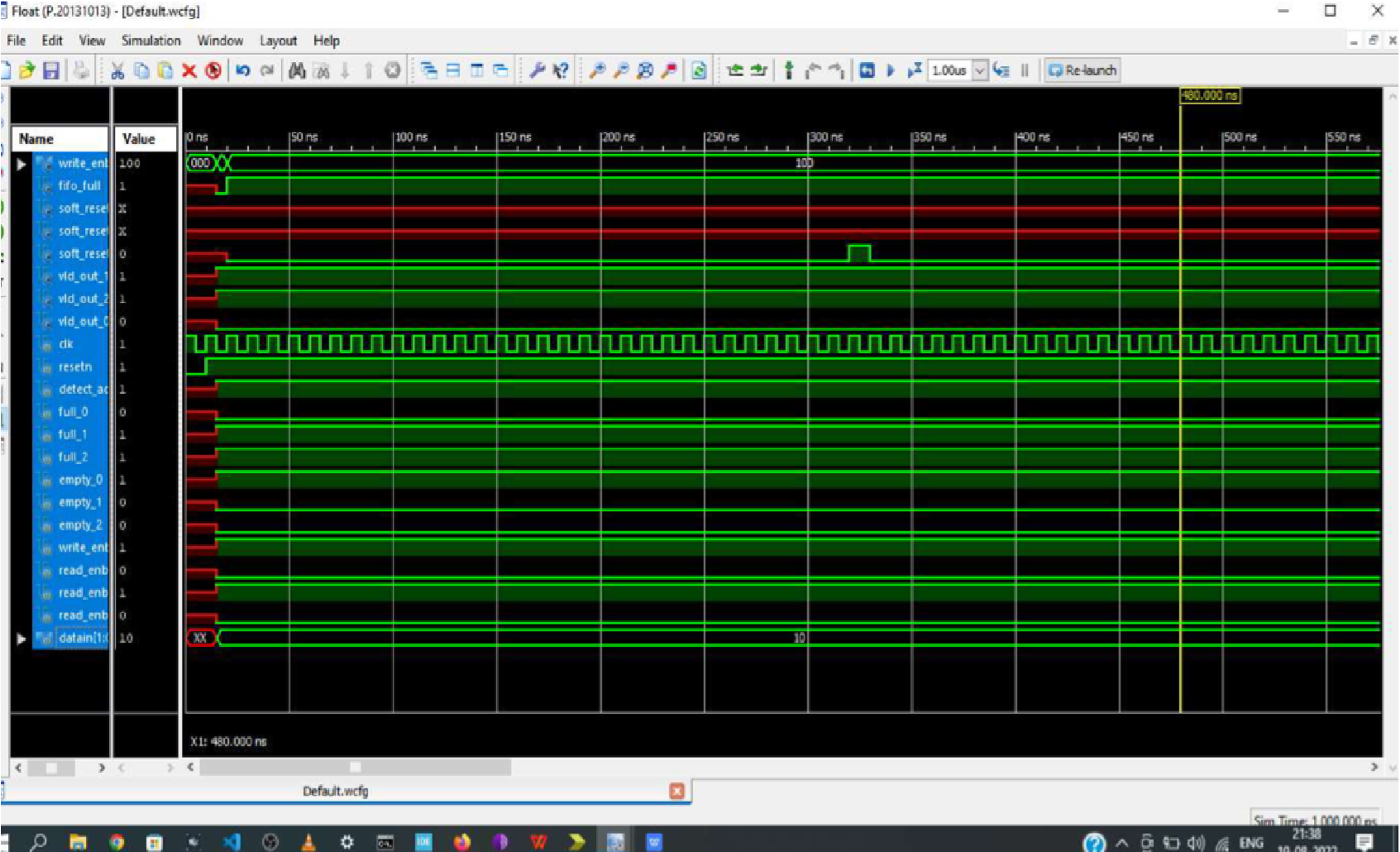
end

endtask

initial

begin

reset; #5; task1(); #1000;

$finish; end

endmodule

## Simulation

3) FSM RTL

module router\_fsm(input clk,resetn,packet\_valid, input [1:0] datain, input

fifo\_full,fifo\_empty\_0,fifo\_empty\_1,fifo\_empty\_2,soft\_reset\_0,soft\_reset\_1,soft\_reset\_

2 ,parity\_done, low\_packet\_valid, output

write\_enb\_reg,detect\_add,ld\_state,laf\_state,lfd\_state,full\_state,rst\_int\_reg,busy);

|  |  |  |
| --- | --- | --- |
| parameter decode\_address = | | 4 'b0001, |
| wait\_till\_empty | = | 4 'b0010, |
| load\_first\_data | = | 4'b0011 , |
| load\_data | = | 4 'b0100, |
| load\_parity | = | 4 'b0101, |
| fifo\_full\_state | = | 4'b0110 , |
| load\_after\_full | = | 4'b0111 , |
| check\_parity\_error | = | 4 'b1000; |

reg [3:0] present\_state, next\_state; reg [1:0] temp;

//temp logic always@(posedge clk) begin if(~resetn) temp<=2'b0;

else if(detect\_add) temp<=datain;

end

// reset logic for states

always@(posedge clk) begin

if(!resetn) present\_state<=decode\_address;

else if (((soft\_reset\_0) && (temp==2'b00)) || ((soft\_reset\_1) && ( temp==2'b01)) || ((soft\_reset\_2) && (temp==2'b 10)))

present\_state<=decode\_address;

else

present\_state<=next\_state;

end

//state machine logic

always@(\*) begin case(present\_state) decode\_address:

begin if((packet\_valid && (datain==2'b00) && fifo\_empty\_0)||

( packet\_valid && (datain==2'b01) && fifo\_empty\_1)|| (packet\_valid && ( datain==2'b10) && fifo\_empty\_ 2))

next\_state<=load\_first\_data; //lfd\_state

else if((packet\_valid && (datain==2'b00) &&

! fifo\_empty\_0)||(packet\_valid && (datain==2'b01) && !fifo\_empty\_1)||(packet\_valid && (datain==2'b10) && !fifo\_empty\_2)) next\_state<=wait\_till\_empty; //wait till empty

state

else next\_state<=decode\_address; // same state

end

load\_first\_data: // load first data state begin next\_state<=load\_data;

end

wait\_till\_empty: //wait till empty state

begin if((fifo\_empty\_0 && (temp==2'b00))||(fifo\_empty\_1 && ( temp==2'b01))||(fifo\_empty\_2 && (temp==2'b10))) //fifo is empty and were using same fifo next\_state<=load\_first\_data;

else next\_state<=wait\_till\_empty;

end

load\_data: //load data

begin if(fifo\_full==1'b1)

next\_state<=fifo\_full\_state;

else begin if (!fifo\_full && !packet\_valid) next\_state<=load\_parity;

else next\_state<=load\_data; end

end

fifo\_full\_state: //fifo full state begin if(fifo\_full==0) next\_state<=load\_after\_full;

else next\_state<=fifo\_full\_state; end

load\_after\_full: // load after full state begin if(!parity\_done && low\_packet\_valid) next\_state<=load\_parity;

else if(!parity\_done && !low\_packet\_valid) next\_state<=load\_data;

else begin if(parity\_done==1'b1) next\_state<=decode\_address;

else next\_state<=load\_after\_full; end

end

load\_parity: // load parity state begin next\_state<=check\_parity\_error;

end

check\_parity\_error: // check parity error

begin

if(!fifo\_full) next\_state<=decode\_address;

else

next\_state<=fifo\_full\_state;

end

default: //default state

next\_state<=decode\_address;

endcase

// state machine completed end

// output logic

assign

busy=((present\_state==load\_first\_data)||(present\_state==load\_parity)||(present\_state==fi fo\_full\_state)||(present\_state==load\_after\_full)||(present\_state==wait\_till\_empty)||(prese nt\_state==check\_parity\_error))?1:0;

assign detect\_add=((present\_state==decode\_address))?1:0; assign lfd\_state=((present\_state==load\_first\_data))?1:0; assign ld\_state=((present\_state==load\_data))?1:0; assign

write\_enb\_reg=((present\_state==load\_data)||(present\_state==load\_after\_full)||(present\_s tate==load\_parity))?1:0;

assign full\_state=((present\_state==fifo\_full\_state))?1:0; assign laf\_state=((present\_state==load\_after\_full))?1:0; assign rst\_int\_reg=((present\_state==check\_parity\_error))?1:0; endmodule

**Testbench** module router\_fsm\_tb();

reg clk,resetn, packet\_valid; reg [1:0]datain;

reg fifo\_full, fifo\_empty\_0, fifo\_empty\_1, fifo\_empty\_2, soft\_reset\_0, soft\_reset\_1, soft\_reset\_2, parity\_done, low\_packet\_valid;

wire write\_enb\_reg, detect\_add, ld\_state, laf\_state, lfd\_state, full\_state, rst\_int\_reg, busy;

router\_fsm DUT ( .clk(clk),

.resetn(resetn),

.packet\_valid(packet\_valid),

.datain(datain),

.fifo\_full(fifo\_full),

.fifo\_empty\_0(fifo\_empty\_0),

.fifo\_empty\_1(fifo\_empty\_1),

.fifo\_empty\_2(fifo\_empty\_2),

.soft\_reset\_0(soft\_reset\_0),

.soft\_reset\_1(soft\_reset\_1),

.soft\_reset\_2(soft\_reset\_2),

.parity\_done(parity\_done),

.low\_packet\_valid(low\_packet\_valid),

.write\_enb\_reg(write\_enb\_reg),

.detect\_add(detect\_add),

.ld\_state(ld\_state),

.laf\_state(laf\_state),

.lfd\_state(lfd\_state),

.full\_state(full\_state),

.rst\_int\_reg(rst\_int\_reg), .busy(busy) );

//clock generation initial begin clk = 1; forever #5 clk=~clk; end

task reset; begin resetn=1'b0; #10; resetn=1'b1;

end

endtask

task task1; begin packet\_valid=1'b0; datain=2'b10; fifo\_full=1'b0; fifo\_empty\_0=1'b0; fifo\_empty\_1=1'b1; fifo\_empty\_2=1'b1; soft\_reset\_0=1'b1; soft\_reset\_1=1'b0; soft\_reset\_2=1'b1; parity\_done =1'b1; low\_packet\_valid=1'b1;

end

endtask

task task2; begin packet\_valid=1'b1; datain=2'b01; fifo\_full=1'b0; fifo\_empty\_0=1'b1; fifo\_empty\_1=1'b0; fifo\_empty\_2=1'b1; soft\_reset\_0=1'b0; soft\_reset\_1=1'b1; soft\_reset\_2=1'b1; parity\_done =1'b1; low\_packet\_valid=1'b1;

end

endtask

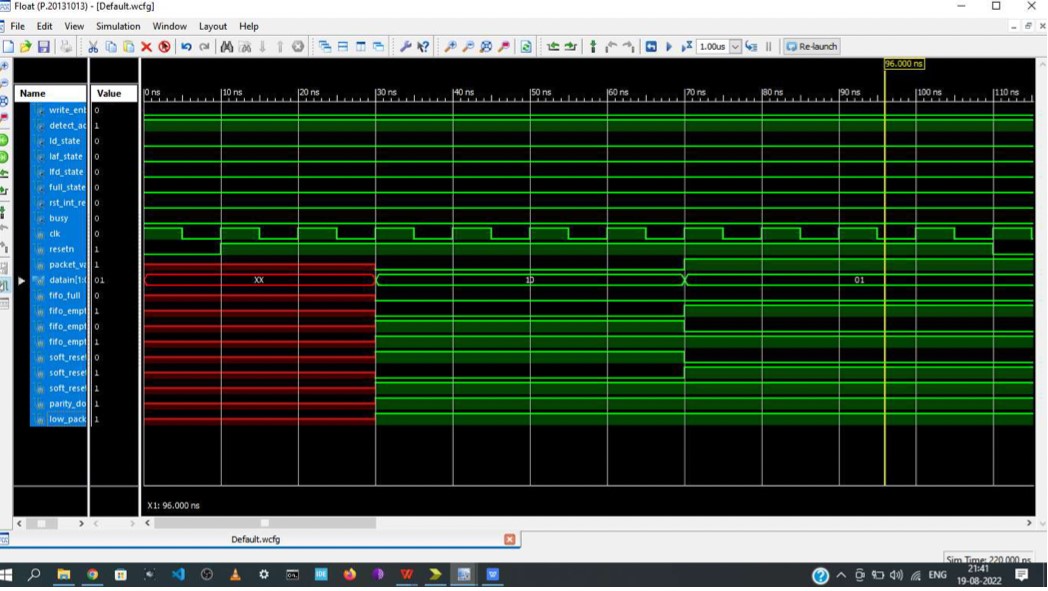
initial

begin

reset; #20; task1; #40; task2; #40;

reset; #100;

$finish; end endmodule **Simulations**



## 4) Register

## RTL

module router\_reg(input clk,resetn,packet\_valid, input [7:0] datain, input fifo\_full,detect\_add,ld\_state,laf\_state,full\_state,lfd\_state,rst\_int\_reg, output reg err,parity\_done,low\_packet\_valid, output reg [7:0] dout); reg [7:0] hold\_header\_byte,fifo\_full\_state\_byte,internal\_parity,packet\_parity\_byte;

//parity done always@(posedge clk) begin

if(!resetn) begin parity\_done<=1'b0;

end

else begin if(ld\_state && !fifo\_full && !packet\_valid) parity\_done<=1'b1;

else if(laf\_state && low\_packet\_valid && !parity\_done) parity\_done<=1'b1;

else begin if(detect\_add) parity\_done<=1'b0;

end

end

end

//low\_packet valid always@(posedge clk) begin

if(!resetn) low\_packet\_valid<=1'b0;

else begin if(rst\_int\_reg) low\_packet\_valid<=1'b0;

if(ld\_state==1'b1 && packet\_valid==1'b0) low\_packet\_valid<=1'b1;

end end

//dout always@(posedge clk)

begin

if(!resetn) dout<=8'b0;

else begin if(detect\_add && packet\_valid) hold\_header\_byte<=datain;

else if(lfd\_state) dout<=hold\_header\_byte;

else if(ld\_state && !fifo\_full) dout<=datain;

else if(ld\_state && fifo\_full) fifo\_full\_state\_byte<=datain;

else begin

if(laf\_state) dout<=fifo\_full\_state\_byte; end end

end

// internal parity

always@(posedge clk) begin

if(!resetn) internal\_parity<=8'b0;

else if(lfd\_state) internal\_parity<=internal\_parity ^ hold\_header\_byte;

else if(ld\_state && packet\_valid && !full\_state) internal\_parity<=internal\_parity ^ datain;

else begin if (detect\_add) internal\_parity<=8'b0; end end

//error and packet\_

always@(posedge clk) begin

if(!resetn) packet\_parity\_byte<=8'b0;

else begin if(!packet\_valid && ld\_state) packet\_parity\_byte<=datain;

end

end

//error

always@(posedge clk) begin

if(!resetn) err<=1'b0;

else begin if(parity\_done) begin if(internal\_parity!=packet\_parity\_byte) err<=1'b1;

else err<=1'b0;

end

end

end endmodule

## Testbench

module router\_reg\_tb();

reg clk, resetn, packet\_valid,fifo\_full, detect\_add, ld\_state, laf\_state, full\_state, lfd\_state, rst\_int\_reg; reg [7:0] datain;

wire err, parity\_done, low\_packet\_valid; wire [7:0]dout; integer i;

router\_reg DUT( .clk(clk),

.resetn(resetn),

.packet\_valid(packet\_valid),

.fifo\_full(fifo\_full),

.detect\_add(detect\_add),

.ld\_state(ld\_state),

.laf\_state(laf\_state),

.full\_state(full\_state),

.lfd\_state(lfd\_state),

.rst\_int\_reg(rst\_int\_reg),

.datain( ),

.err(err),

.parity\_done(parity\_done),

.low\_packet\_valid(low\_packet\_valid),

.dout(dout));

//clock generation

initial

begin clk = 1; forever #5 clk=~clk; end

task reset; begin resetn=1'b0; #10; resetn=1'b1;

end

endtask

task packet1();

reg [7:0]header, payload\_data, parity; reg [5:0]payloadlen; begin

@(negedge clk); payloadlen=8; parity=0; detect\_add=1'b1; packet\_valid=1'b1; header={payloadlen,2'b10}; datain=header; parity=parity^datain; @(negedge clk); detect\_add=1'b0; lfd\_state=1'b1;

for(i=0;i<payloadlen;i=i+1) begin @(negedge clk); lfd\_state=0; ld\_state=1;

payload\_data={$random}%256; datain=payload\_data;

parity=parity^datain; end

@(negedge clk); packet\_valid=0; datain=parity;

@(negedge clk);

ld\_state=0; end

endtask task packet2();

reg [7:0]header, payload\_data, parity; reg [5:0]payloadlen; begin

@(negedge clk); payloadlen=8; parity=0; detect\_add=1'b1; packet\_valid=1'b1; header={payloadlen,2'b10}; datain=header; parity=parity^datain;

@(negedge clk); detect\_add=1'b0; lfd\_state=1'b1; for(i=0;i<payloadlen;i=i+1) begin @(negedge clk); lfd\_state=0; ld\_state=1;

payload\_data={$random}%256; datain=payload\_data;

parity=parity^datain; end

@(negedge clk); packet\_valid=0; datain=!parity;

@(negedge clk);

ld\_state=0; end

endtask

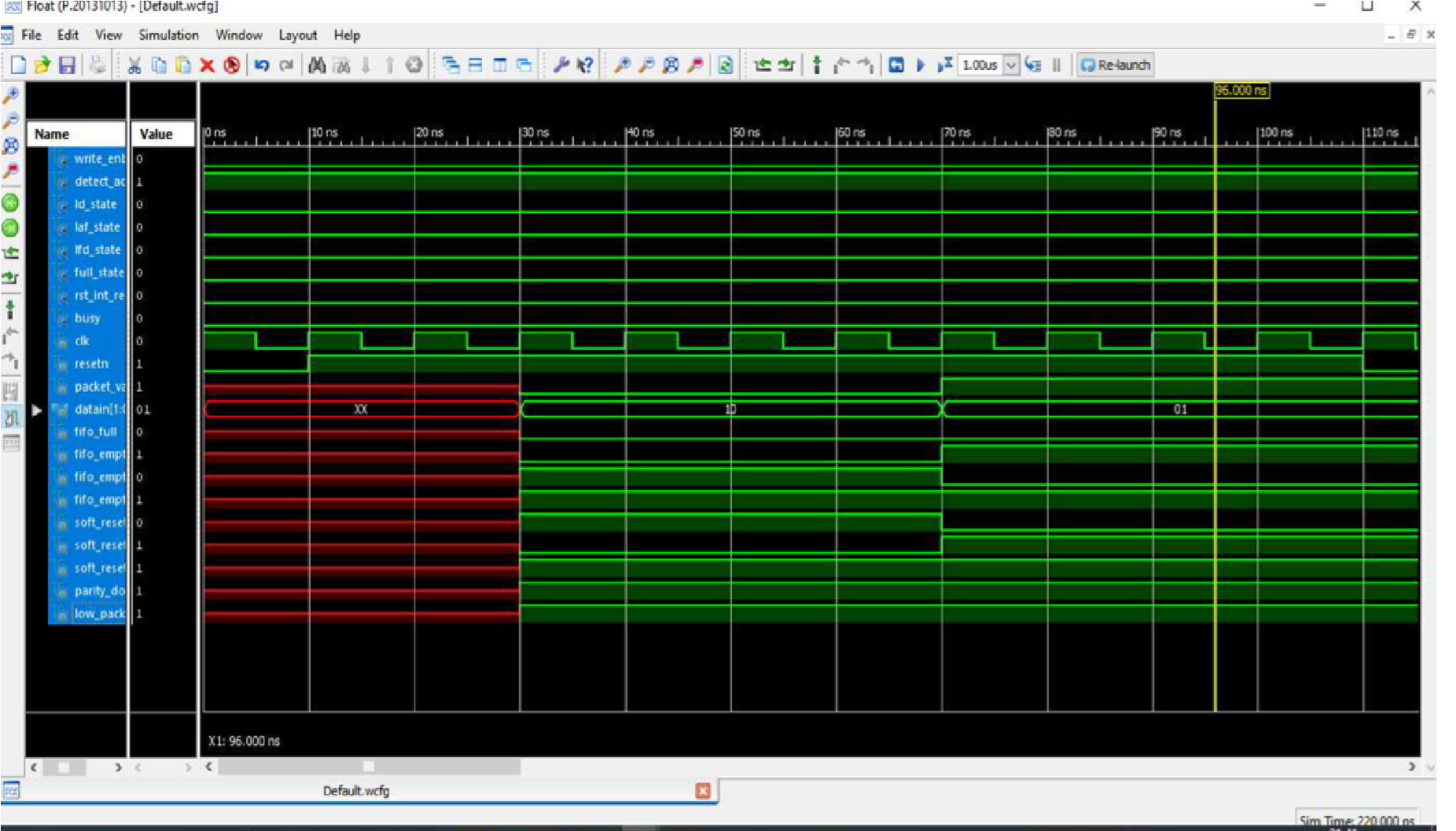
initial

begin

reset; fifo\_full=1'b0; laf\_state=1'b0;

full\_state=1'b0; #20; packet1(); #105; packet2();

$finish; end endmodule **Simulation**



## 4 . Student Feedback

Learning is the process of life and being student of this trust is so special part of my life. Thanks to all teachers who guided me in this course and in extra curricular activities. Special thanks to my trainer who taught this topic so easily. This experience will help me grow my future in every direction and serve the needs of society as best as possible.

## 5. Uniqueness of the Course

This course teaches in depth of Verilog where institutions fail to teach. I have gained so much confidence while asking doubts to know answers in depth because the trainer is friendly and supportive.

## 6. Concluding Remarks

Sure trust is not just a trust but it’s a family where each and every student is getting help in every manner. Being member of this trust i have tried to contribute in every manner. Students from different places meet here and help each other. In future i will do my level best in growing this trust.